**SUPERSET ID:** **6376594**

**DESIGN PATTERN AND PRINCIPLES(HANDS-ON)**

**IDE:ECLIPSE**

**Exercise 1: Implementing the Singleton Pattern**

**Project Name: SingletonPatternExample**

Logger.java Code:

**package** SingletonPatternExample;

**public** **class** Logger {

**private** **static** Logger *instance*;

**private** Logger() {

System.***out***.println("Logger initialized.");

}

**public** **static** Logger getInstance() {

**if** (*instance* == **null**) {

*instance* = **new** Logger();

}

**return** *instance*;

}

**public** **void** log(String message) {

System.***out***.println("Log: " + message);

}

}

Main.java Code:

**package** SingletonPatternExample;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Logger logger1 = Logger.*getInstance*();

logger1.log("First message");

Logger logger2 = Logger.*getInstance*();

logger2.log("Second message");

**if** (logger1 == logger2) {

System.***out***.println("Both logger instances are the same (singleton works).");

} **else** {

System.***out***.println("Logger instances are different (singleton failed).");

}

}

}

Expected Output:

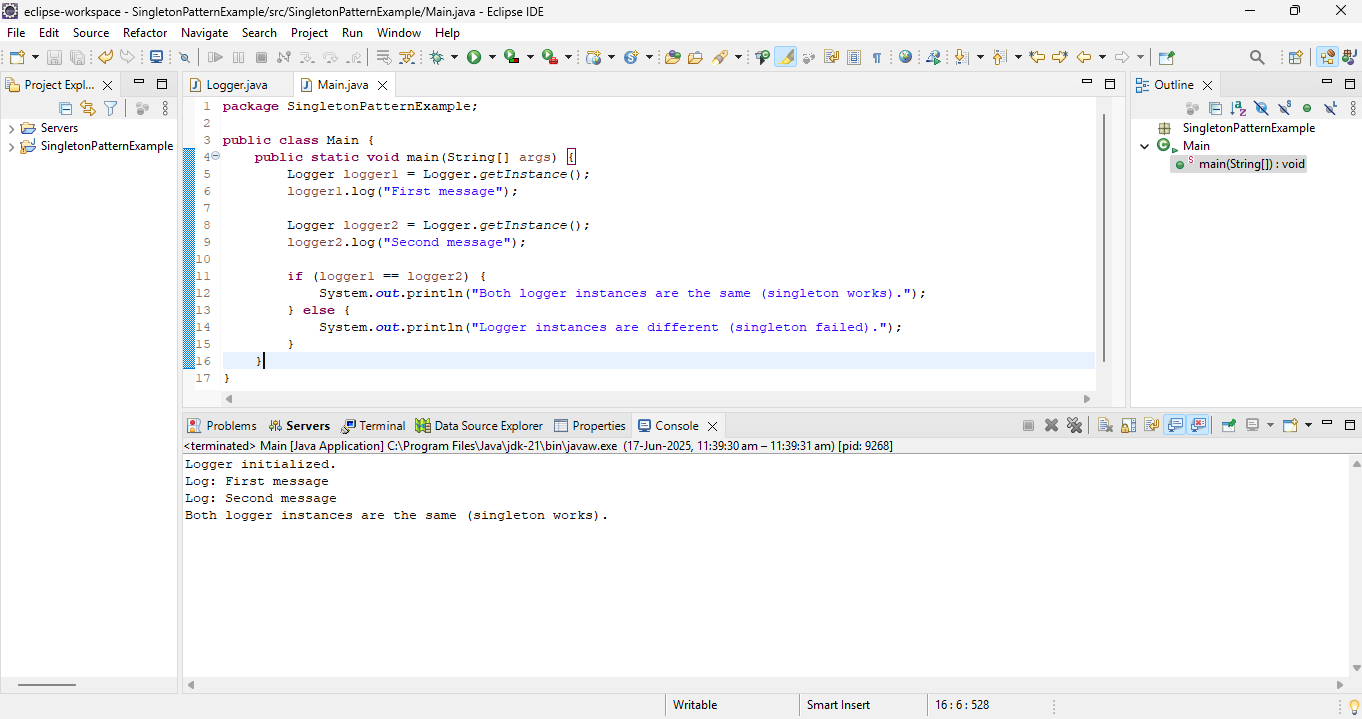
Logger initialized.

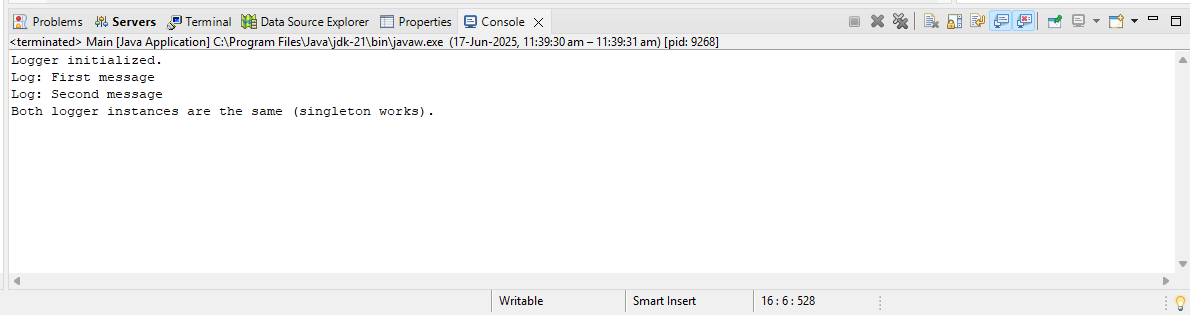
Log: First message

Log: Second message

Both logger instances are the same (singleton works).

Output:





**Exercise 2: Implementing the Factory Method Pattern**

**Project Name**: **FactoryMethodPatternExample**

WordDocument.java Code:

**package** FactoryMethodPatternExample;

**public** **class** WordDocument **implements** Document {

**public** **void** open() {

System.***out***.println("Opening a Word document.");

}

}

PdfDocument.java Code:

**package** FactoryMethodPatternExample;

**public** **class** PdfDocument **implements** Document {

**public** **void** open() {

System.***out***.println("Opening a PDF document.");

}

}

ExcelDocument.java Code:

**package** FactoryMethodPatternExample;

**public** **class** ExcelDocument **implements** Document {

**public** **void** open() {

System.***out***.println("Opening an Excel document.");

}

}

DocumentFactory.java Code:

**package** FactoryMethodPatternExample;

**public** **interface** Document {

**void** open();

}

**package** FactoryMethodPatternExample;

**public** **abstract** **class** DocumentFactory {

**public** **abstract** Document createDocument();

}

WordDocumentFactory.java Code:

**package** FactoryMethodPatternExample;

**public** **class** WordDocumentFactory **extends** DocumentFactory {

**public** Document createDocument() {

**return** **new** WordDocument();

}

}

PdfDocumentFactory.java Code:

**package** FactoryMethodPatternExample;

**public** **class** PdfDocumentFactory **extends** DocumentFactory {

**public** Document createDocument() {

**return** **new** PdfDocument();

}

}

ExcelDocumentFactory.java Code:

**package** FactoryMethodPatternExample;

**public** **class** PdfDocumentFactory **extends** DocumentFactory {

**public** Document createDocument() {

**return** **new** PdfDocument();

}

}

Main.java Code:

**package** FactoryMethodPatternExample;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

DocumentFactory wordFactory = **new** WordDocumentFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = **new** PdfDocumentFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

DocumentFactory excelFactory = **new** ExcelDocumentFactory();

Document excel = excelFactory.createDocument();

excel.open();

}

}

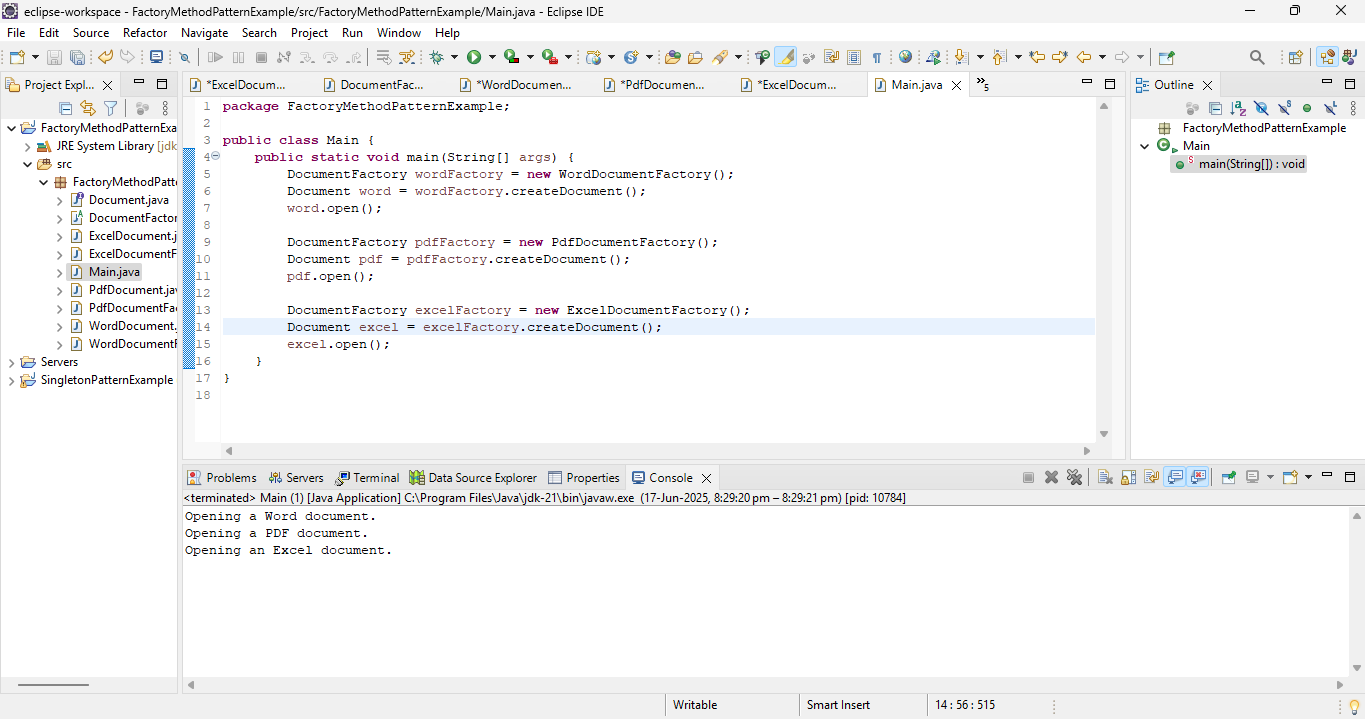
Expected Output:

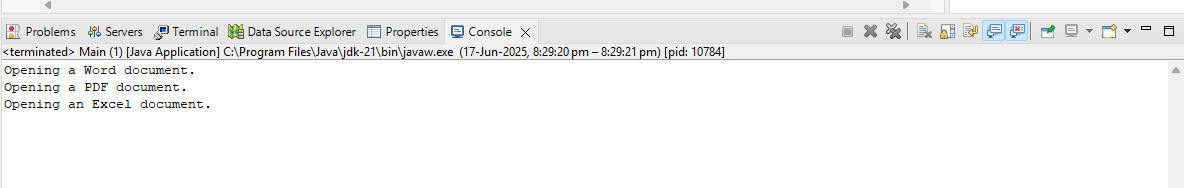
Opening a Word document.

Opening a PDF document.

Opening an Excel document.

Output:





**Exercise 3: Implementing the Builder Pattern**

**Project Name:** **BuilderPatternExample**

Computer.java Code:

**package** BuilderPatternExample;

**public** **class** Computer {

**private** String CPU;

**private** String RAM;

**private** String storage;

**private** **boolean** hasGraphicsCard;

**private** **boolean** hasWiFi;

**private** Computer(Builder builder) {

**this**.CPU = builder.CPU;

**this**.RAM = builder.RAM;

**this**.storage = builder.storage;

**this**.hasGraphicsCard = builder.hasGraphicsCard;

**this**.hasWiFi = builder.hasWiFi;

}

**public** **static** **class** Builder {

**private** String CPU;

**private** String RAM;

**private** String storage;

**private** **boolean** hasGraphicsCard;

**private** **boolean** hasWiFi;

**public** Builder setCPU(String CPU) {

**this**.CPU = CPU;

**return** **this**;

}

**public** Builder setRAM(String RAM) {

**this**.RAM = RAM;

**return** **this**;

}

**public** Builder setStorage(String storage) {

**this**.storage = storage;

**return** **this**;

}

**public** Builder setGraphicsCard(**boolean** hasGraphicsCard) {

**this**.hasGraphicsCard = hasGraphicsCard;

**return** **this**;

}

**public** Builder setWiFi(**boolean** hasWiFi) {

**this**.hasWiFi = hasWiFi;

**return** **this**;

}

**public** Computer build() {

**return** **new** Computer(**this**);

}

}

**public** **void** showConfig() {

System.***out***.println("CPU: " + CPU);

System.***out***.println("RAM: " + RAM);

System.***out***.println("Storage: " + storage);

System.***out***.println("Graphics Card: " + (hasGraphicsCard ? "Yes" : "No"));

System.***out***.println("WiFi: " + (hasWiFi ? "Yes" : "No"));

}

}

Main.java Code:

**package** BuilderPatternExample;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Computer basicComputer = **new** Computer.Builder()

.setCPU("Intel i3")

.setRAM("8GB")

.setStorage("256GB SSD")

.build();

System.***out***.println("Basic Computer Configuration:");

basicComputer.showConfig();

System.***out***.println();

Computer gamingComputer = **new** Computer.Builder()

.setCPU("Intel i9")

.setRAM("32GB")

.setStorage("1TB SSD")

.setGraphicsCard(**true**)

.setWiFi(**true**)

.build();

System.***out***.println("Gaming Computer Configuration:");

gamingComputer.showConfig();

}

}

Expected Output:

CPU: Intel i3

RAM: 8GB

Storage: 256GB SSD

Graphics Card: No

WiFi: No

Gaming Computer Configuration:

CPU: Intel i9

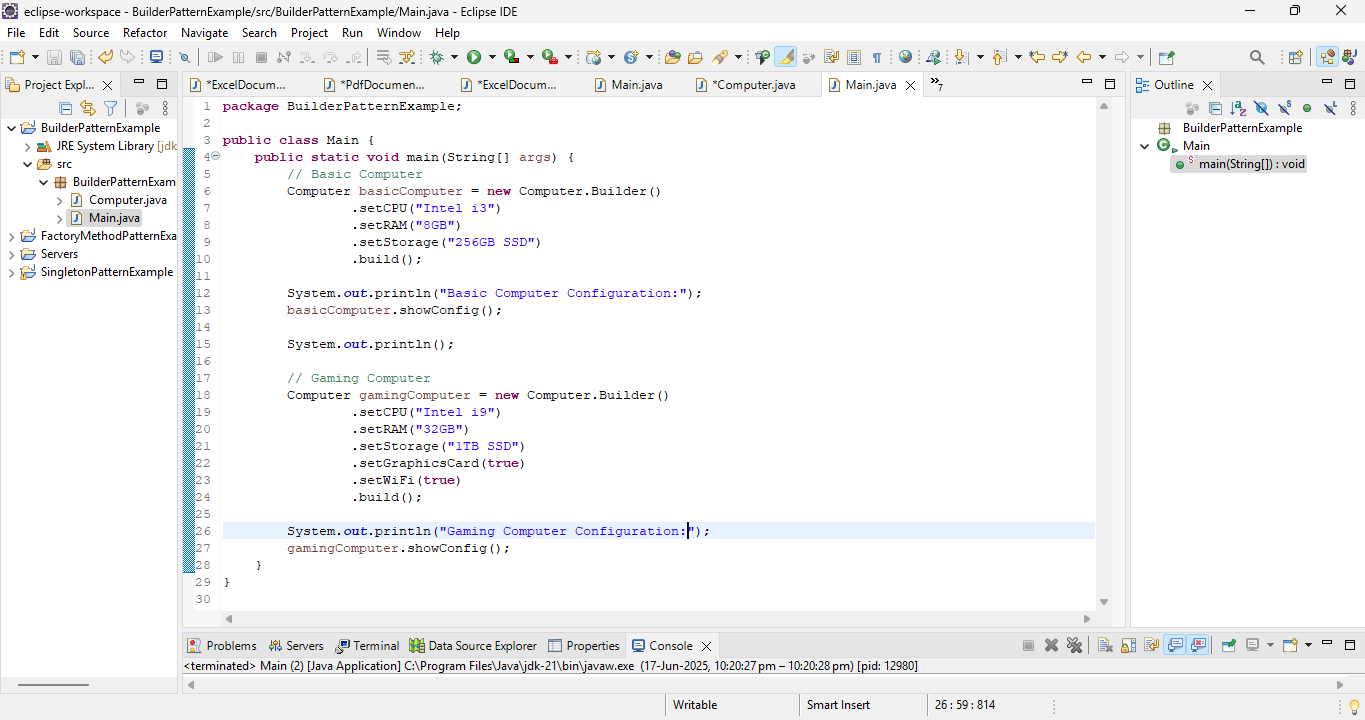
RAM: 32GB

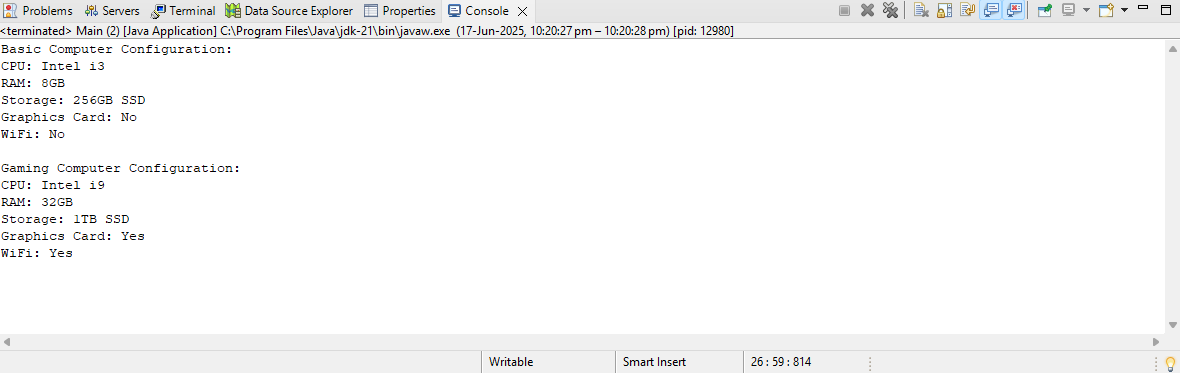
Storage: 1TB SSD

Graphics Card: Yes

WiFi: Yes

Output:





**Exercise 4: Implementing the Adapter Pattern**

**Project Name:** **AdapterPatternExample**

PaymentProcessor.java Code:

**package** AdapterPatternExample;

**public** **interface** PaymentProcessor {

**void** processPayment(**double** amount);

}

PayPalPayment.java Code:

**package** AdapterPatternExample;

**public** **class** PayPalPayment {

**public** **void** sendPayment(**double** amount) {

System.***out***.println("Paid $" + amount + " using PayPal.");

}

}

StripePayment.java Code:

**package** AdapterPatternExample;

**public** **class** StripePayment {

**public** **void** makeStripePayment(**double** amount) {

System.***out***.println("Paid $" + amount + " using Stripe.");

}

}

PayPalAdapter.java Code:

**package** AdapterPatternExample;

**public** **class** PayPalAdapter **implements** PaymentProcessor {

**private** PayPalPayment payPalPayment;

**public** PayPalAdapter(PayPalPayment payPalPayment) {

**this**.payPalPayment = payPalPayment;

}

**public** **void** processPayment(**double** amount) {

payPalPayment.sendPayment(amount);

}

}

StripeAdapter.java Code:

**package** AdapterPatternExample;

**public** **class** StripeAdapter **implements** PaymentProcessor {

**private** StripePayment stripePayment;

**public** StripeAdapter(StripePayment stripePayment) {

**this**.stripePayment = stripePayment;

}

**public** **void** processPayment(**double** amount) {

stripePayment.makeStripePayment(amount);

}

}

Main.java Code:

**package** AdapterPatternExample;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

PayPalPayment paypal = **new** PayPalPayment();

PaymentProcessor paypalProcessor = **new** PayPalAdapter(paypal);

paypalProcessor.processPayment(250.00);

StripePayment stripe = **new** StripePayment();

PaymentProcessor stripeProcessor = **new** StripeAdapter(stripe);

stripeProcessor.processPayment(450.00);

}

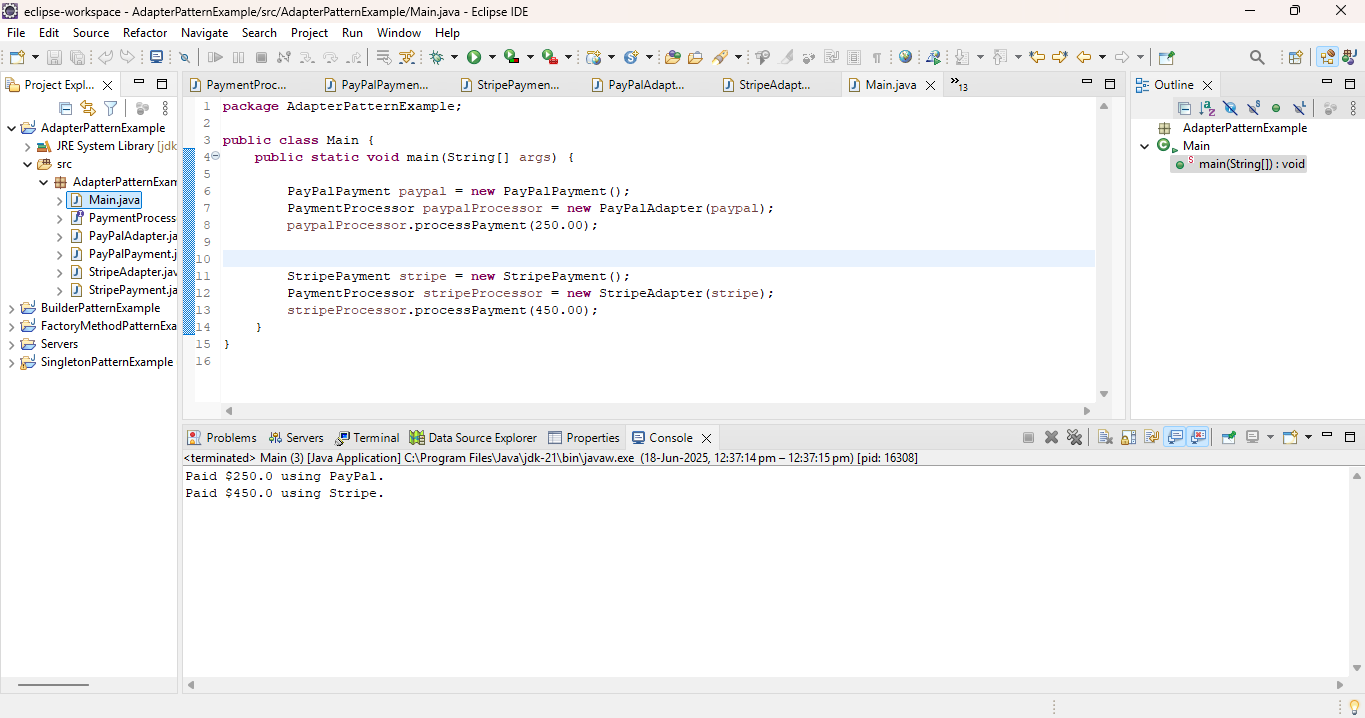
}

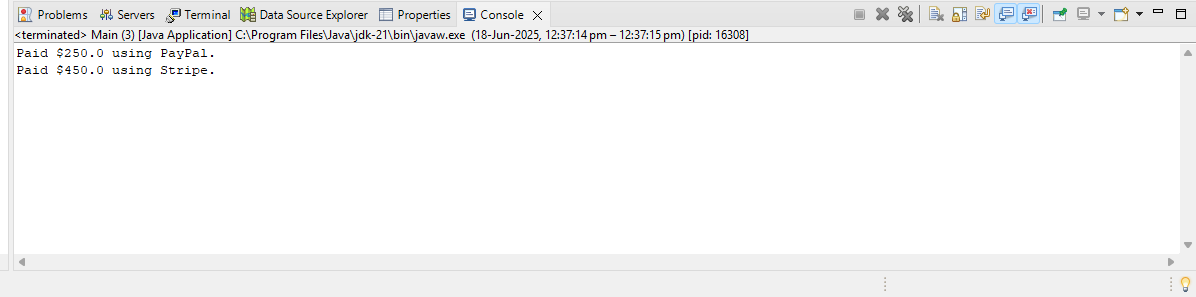
Expected Output:

Paid $250.0 using PayPal.

Paid $450.0 using Stripe.

Output:





**Exercise 5: Implementing the Decorator Pattern**

**Project Name:** **DecoratorPatternExample**

Notifier.java Code:

**package** DecoratorPatternExample;

**public** **interface** Notifier {

**void** send(String message);

}

EmailNotifier.java Code:

**package** DecoratorPatternExample;

**public** **class** EmailNotifier **implements** Notifier {

**public** **void** send(String message) {

System.***out***.println("Sending Email: " + message);

}

}

NotifierDecorator.java Code:

**package** DecoratorPatternExample;

**public** **abstract** **class** NotifierDecorator **implements** Notifier {

**protected** Notifier wrappedNotifier;

**public** NotifierDecorator(Notifier notifier) {

**this**.wrappedNotifier = notifier;

}

**public** **void** send(String message) {

wrappedNotifier.send(message);

}

}

SMSNotifierDecorator.java Code:

**package** DecoratorPatternExample;

**public** **class** SMSNotifierDecorator **extends** NotifierDecorator {

**public** SMSNotifierDecorator(Notifier notifier) {

**super**(notifier);

}

**public** **void** send(String message) {

**super**.send(message);

System.***out***.println("Sending SMS: " + message);

}

}

SlackNotifierDecorator.java Code:

**package** DecoratorPatternExample;

**public** **class** SlackNotifierDecorator **extends** NotifierDecorator {

**public** SlackNotifierDecorator(Notifier notifier) {

**super**(notifier);

}

**public** **void** send(String message) {

**super**.send(message);

System.***out***.println("Sending Slack: " + message);

}

}

Main.java Code:

**package** DecoratorPatternExample;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Notifier emailNotifier = **new** EmailNotifier();

Notifier emailAndSMSNotifier = **new** SMSNotifierDecorator(emailNotifier);

Notifier fullNotifier = **new** SlackNotifierDecorator(emailAndSMSNotifier);

fullNotifier.send("Your order has been shipped!");

}

}

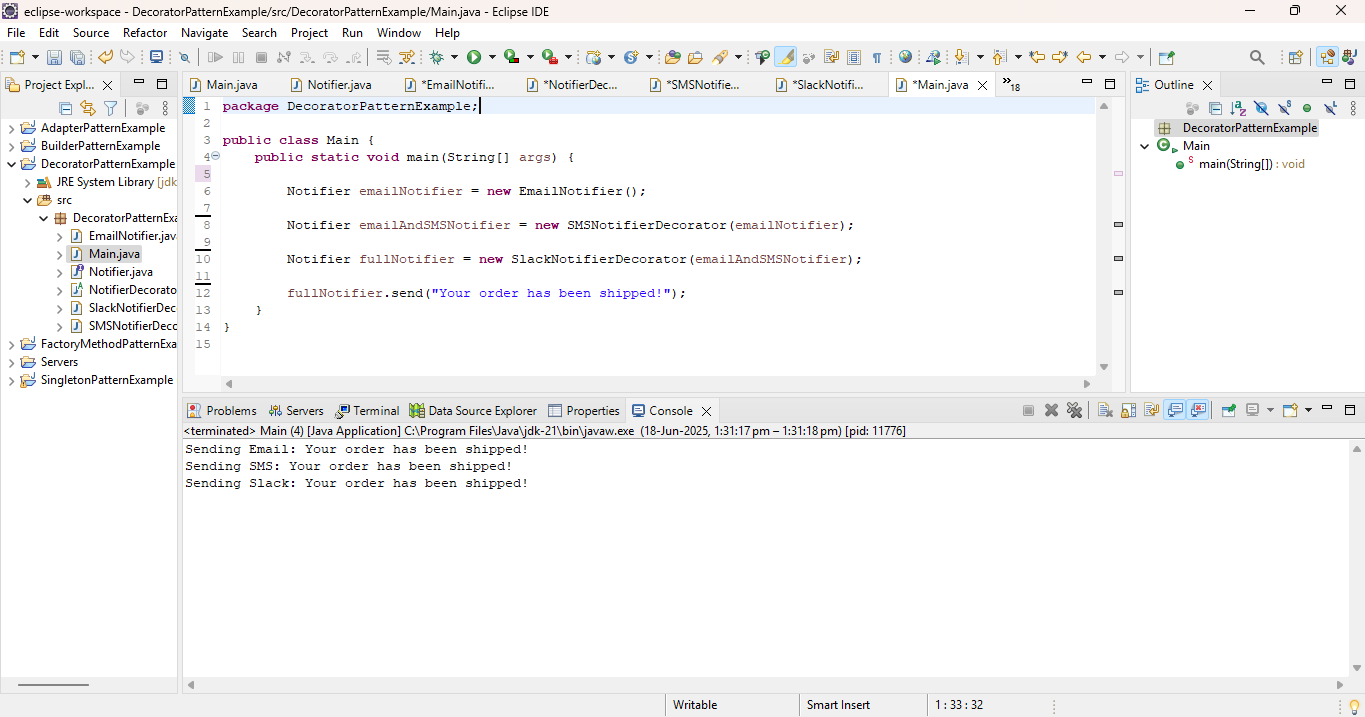
Expected Output:

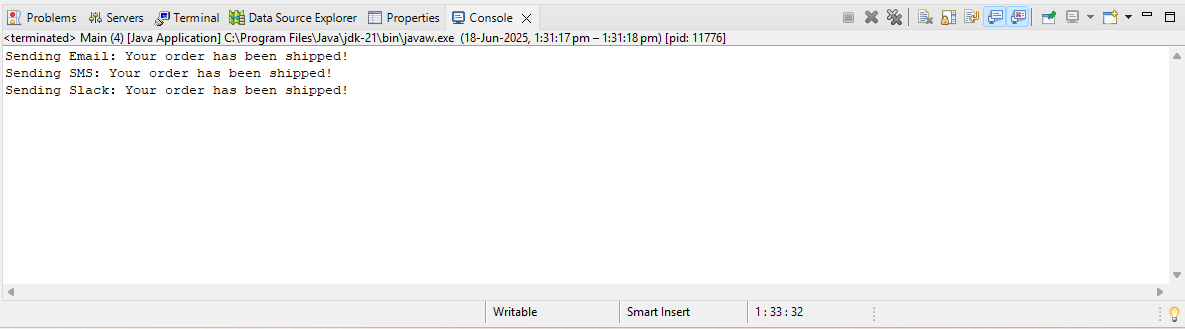
Sending Email: Your order has been shipped!

Sending SMS: Your order has been shipped!

Sending Slack: Your order has been shipped!

Output:





**Exercise 6: Implementing the Proxy Pattern**

**Project Name:** **ProxyPatternExample**

Image.java Code:

**package** ProxyPatternExample;

**public** **interface** Image {

**void** display();

}

RealImage.java Code:

**package** ProxyPatternExample;

**public** **class** RealImage **implements** Image {

**private** String filename;

**public** RealImage(String filename) {

**this**.filename = filename;

loadFromDisk(filename);

}

**private** **void** loadFromDisk(String filename) {

System.***out***.println("Loading image from remote server: " + filename);

}

**public** **void** display() {

System.***out***.println("Displaying image: " + filename);

}

}

ProxyImage.java Code:

**package** ProxyPatternExample;

**public** **class** ProxyImage **implements** Image {

**private** RealImage realImage;

**private** String filename;

**public** ProxyImage(String filename) {

**this**.filename = filename;

}

**public** **void** display() {

**if** (realImage == **null**) {

realImage = **new** RealImage(filename); // Lazy Initialization

} **else** {

System.***out***.println("Image loaded from cache: " + filename);

}

realImage.display();

}

}

Main.java Code:

**package** ProxyPatternExample;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Image image1 = **new** ProxyImage("nature.jpg");

Image image2 = **new** ProxyImage("wildlife.jpg");

image1.display();

image1.display();

image2.display();

image2.display();

}

}

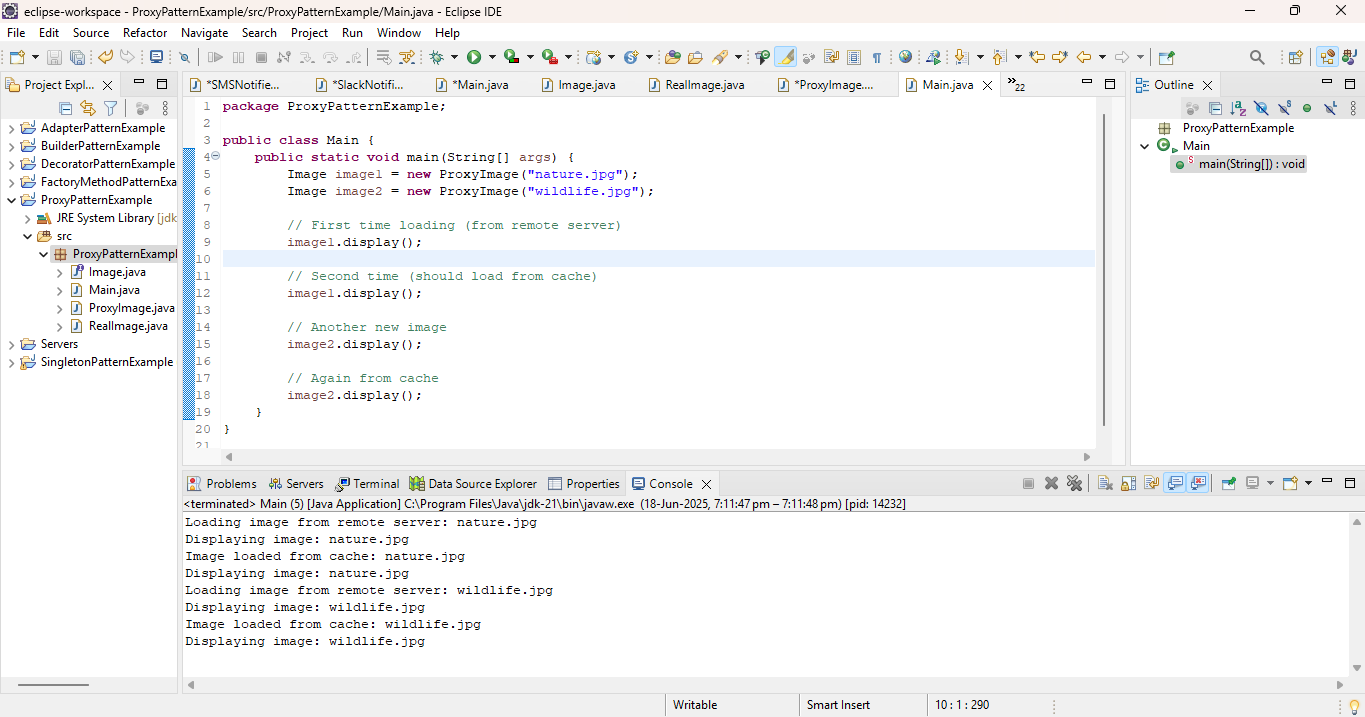
Expected Output:

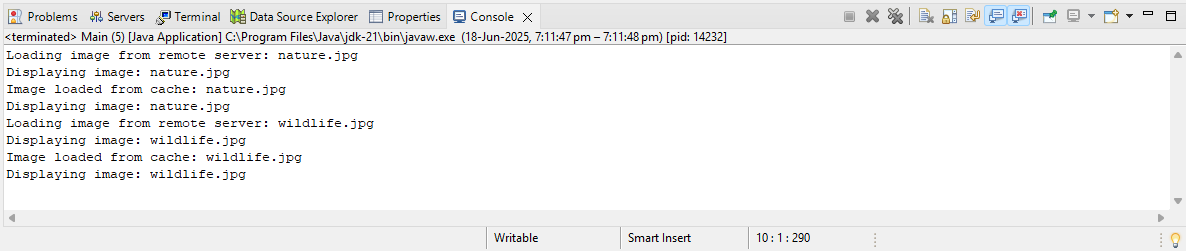
Sending Email: Your order has been shipped!

Sending SMS: Your order has been shipped!

Sending Slack: Your order has been shipped!

Output:





**Exercise 7: Implementing the Observer Pattern**

**Project Name: ObserverPatternExample**

Stock.java Code:

**package** ObserverPatternExample;

**public** **interface** Stock {

**void** registerObserver(Observer o);

**void** removeObserver(Observer o);

**void** notifyObservers();

}

Observer.java.Code:

**package** ObserverPatternExample;

**public** **interface** Observer {

**void** update(**float** price);

}

StockMarket.java Code:

**package** ObserverPatternExample;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** StockMarket **implements** Stock {

**private** List<Observer> observers;

**private** **float** stockPrice;

**public** StockMarket() {

observers = **new** ArrayList<>();

}

**public** **void** registerObserver(Observer o) {

observers.add(o);

}

**public** **void** removeObserver(Observer o) {

observers.remove(o);

}

**public** **void** notifyObservers() {

**for** (Observer o : observers) {

o.update(stockPrice);

}

}

**public** **void** setStockPrice(**float** price) {

**this**.stockPrice = price;

notifyObservers();

}

}

MobileApp.java Code:

**package** ObserverPatternExample;

**public** **class** MobileApp **implements** Observer {

**private** String name;

**public** MobileApp(String name) {

**this**.name = name;

}

**public** **void** update(**float** price) {

System.***out***.println("Mobile App " + name + " received stock update: $" + price);

}

}

WebApp.java Code:

**package** ObserverPatternExample;

**public** **class** WebApp **implements** Observer {

**private** String name;

**public** WebApp(String name) {

**this**.name = name;

}

**public** **void** update(**float** price) {

System.***out***.println("Web App " + name + " received stock update: $" + price);

}

}

Main.java Code:

**package** ObserverPatternExample;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

StockMarket stockMarket = **new** StockMarket();

Observer mobile1 = **new** MobileApp("Investor1");

Observer mobile2 = **new** MobileApp("Investor2");

Observer web1 = **new** WebApp("Dashboard");

stockMarket.registerObserver(mobile1);

stockMarket.registerObserver(mobile2);

stockMarket.registerObserver(web1);

stockMarket.setStockPrice(110.25f);

System.***out***.println("---------------------------");

stockMarket.setStockPrice(122.75f);

stockMarket.removeObserver(mobile2);

System.***out***.println("---------------------------");

stockMarket.setStockPrice(130.00f);

}

}

Expected Output:

Mobile App Investor1 received stock update: $110.25

Mobile App Investor2 received stock update: $110.25

Web App Dashboard received stock update: $110.25

---------------------------

Mobile App Investor1 received stock update: $122.75

Mobile App Investor2 received stock update: $122.75

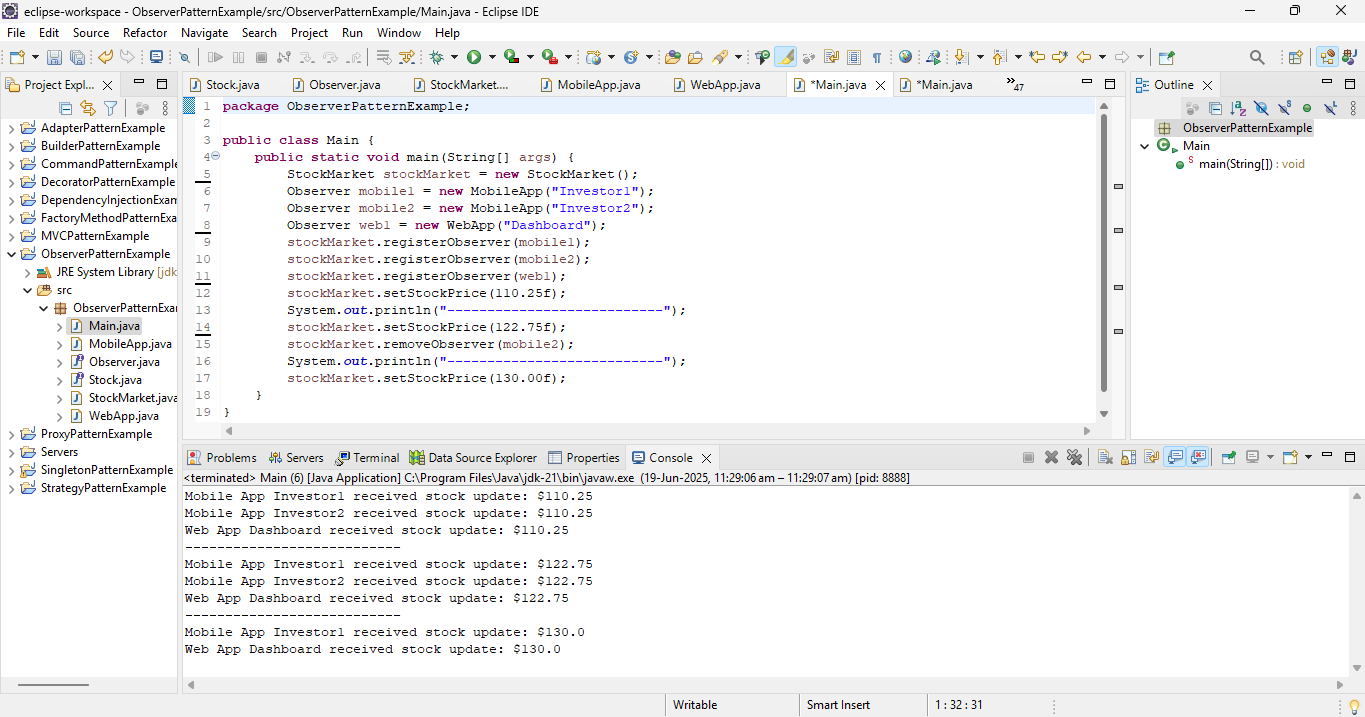
Web App Dashboard received stock update: $122.75

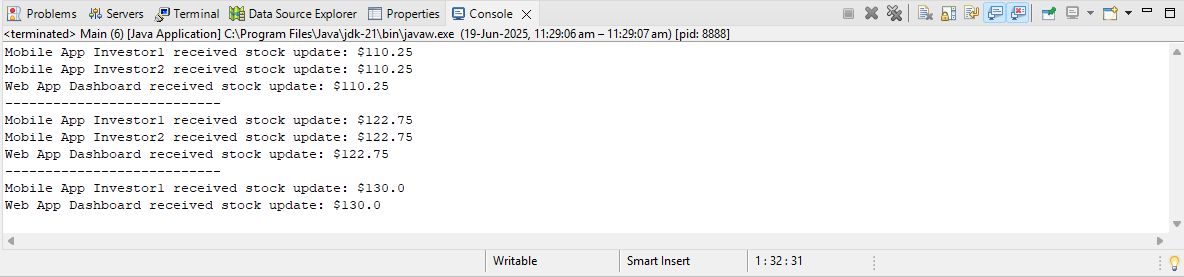
---------------------------

Mobile App Investor1 received stock update: $130.0

Web App Dashboard received stock update: $130.0

Output:





**Exercise 8: Implementing the Strategy Pattern**

**Project Name:** **StrategyPatternExample**

PaymentStrategy.java Code:

**package** StrategyPatternExample;

**public** **interface** PaymentStrategy {

**void** pay(**double** amount);

}

CreditCardPayment.java Code:

**package** StrategyPatternExample;

**public** **class** CreditCardPayment **implements** PaymentStrategy {

**private** String cardNumber;

**public** CreditCardPayment(String cardNumber) {

**this**.cardNumber = cardNumber;

}

**public** **void** pay(**double** amount) {

System.***out***.println("Paid $" + amount + " using Credit Card: " + cardNumber);

}

}

PayPalPayment.java Code:

**package** StrategyPatternExample;

**public** **class** PayPalPayment **implements** PaymentStrategy {

**private** String email;

**public** PayPalPayment(String email) {

**this**.email = email;

}

**public** **void** pay(**double** amount) {

System.***out***.println("Paid $" + amount + " using PayPal account: " + email);

}

}

PaymentContext.java Code:

**package** StrategyPatternExample;

**public** **class** PaymentContext {

**private** PaymentStrategy paymentStrategy;

**public** **void** setPaymentStrategy(PaymentStrategy strategy) {

**this**.paymentStrategy = strategy;

}

**public** **void** payAmount(**double** amount) {

**if** (paymentStrategy == **null**) {

System.***out***.println("Payment method not set.");

} **else** {

paymentStrategy.pay(amount);

}

}

}

Main.java Code:

**package** StrategyPatternExample;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

PaymentContext context = **new** PaymentContext();

PaymentStrategy paypal = **new** PayPalPayment("user@example.com");

context.setPaymentStrategy(paypal);

context.payAmount(500.00);

System.***out***.println("--------------------------");

PaymentStrategy creditCard = **new** CreditCardPayment("1234-5678-9876-5432");

context.setPaymentStrategy(creditCard);

context.payAmount(1500.00);

}

}

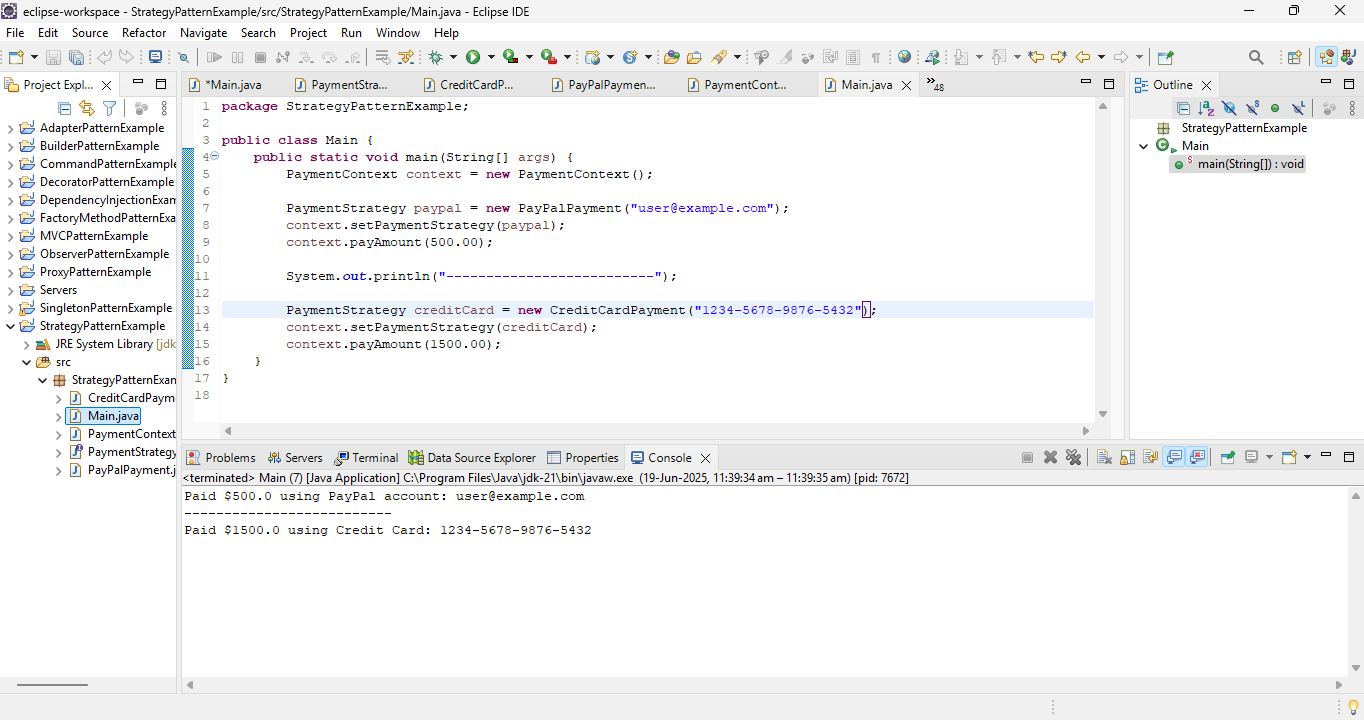
Expected Output:

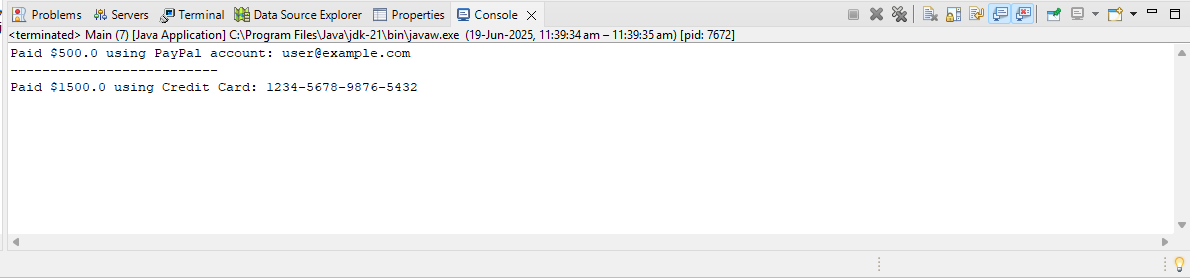
Paid $500.0 using PayPal account: user@example.com

--------------------------

Paid $1500.0 using Credit Card: 1234-5678-9876-5432

Output:





**Exercise 9: Implementing the Command Pattern**

**Project Name:** **CommandPatternExample**

Command.java Code:

**package** CommandPatternExample;

**public** **interface** Command {

**void** execute();

}

Light.java Code:

**package** CommandPatternExample;

**public** **class** Light {

**public** **void** turnOn() {

System.***out***.println("Light is ON");

}

**public** **void** turnOff() {

System.***out***.println("Light is OFF");

}

}

LightOnCommand.java Code:

**package** CommandPatternExample;

**public** **class** LightOnCommand **implements** Command {

**private** Light light;

**public** LightOnCommand(Light light) {

**this**.light = light;

}

**public** **void** execute() {

light.turnOn();

}

}

LightOffCommand.java Code:

**package** CommandPatternExample;

**public** **class** LightOffCommand **implements** Command {

**private** Light light;

**public** LightOffCommand(Light light) {

**this**.light = light;

}

**public** **void** execute() {

light.turnOff();

}

}

RemoteControl.java Code:

**package** CommandPatternExample;

**public** **class** RemoteControl {

**private** Command command;

**public** **void** setCommand(Command command) {

**this**.command = command;

}

**public** **void** pressButton() {

command.execute();

}

}

Main.java Code:

**package** CommandPatternExample;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Light livingRoomLight = **new** Light();

Command lightOn = **new** LightOnCommand(livingRoomLight);

Command lightOff = **new** LightOffCommand(livingRoomLight);

RemoteControl remote = **new** RemoteControl();

remote.setCommand(lightOn);

remote.pressButton();

remote.setCommand(lightOff);

remote.pressButton();

}

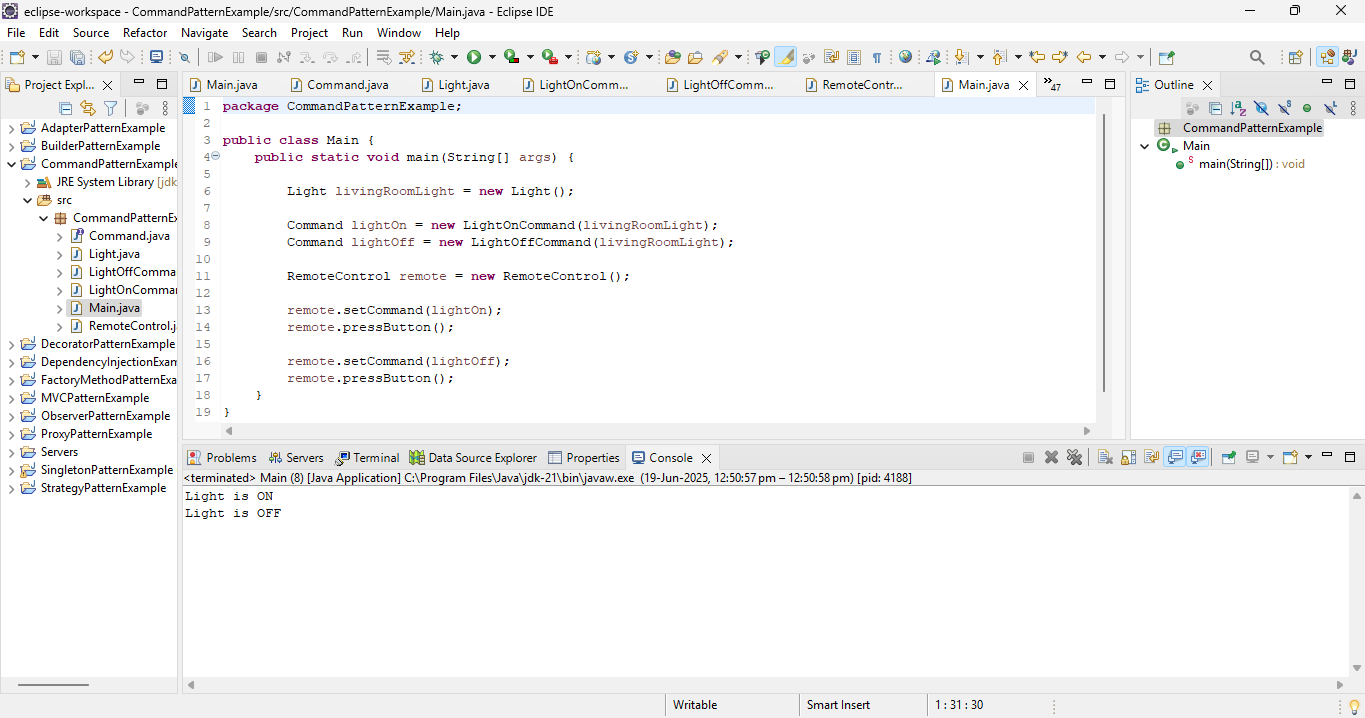
}

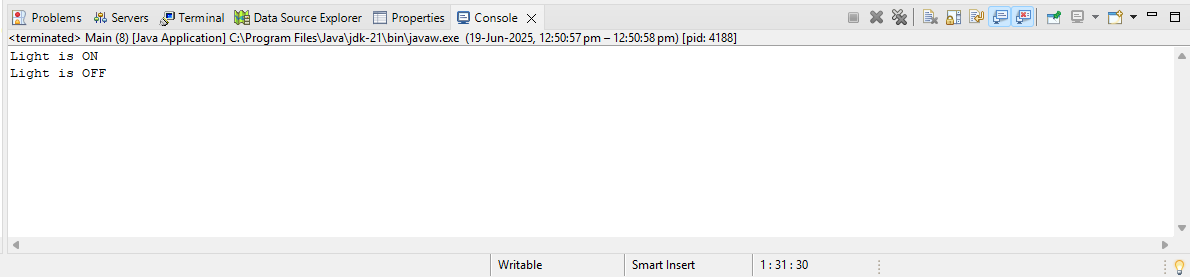
Expected Output:

Light is ON

Light is OFF

Output:





**Exercise 10: Implementing the MVC Pattern**

**Project Name:** **MVCPatternExample**

Student.java Code:

**package** MVCPatternExample;

**public** **class** Student {

**private** String name;

**private** **int** id;

**private** **double** grade;

**public** Student(String name, **int** id, **double** grade) {

**this**.name = name;

**this**.id = id;

**this**.grade = grade;

}

**public** String getName() { **return** name; }

**public** **void** setName(String name) { **this**.name = name; }

**public** **int** getId() { **return** id; }

**public** **void** setId(**int** id) { **this**.id = id; }

**public** **double** getGrade() { **return** grade; }

**public** **void** setGrade(**double** grade) { **this**.grade = grade; }

}

StudentView.java Code:

**package** MVCPatternExample;

**public** **class** StudentView {

**public** **void** displayStudentDetails(String name, **int** id, **double** grade) {

System.***out***.println("Student Details:");

System.***out***.println("Name: " + name);

System.***out***.println("ID: " + id);

System.***out***.println("Grade: " + grade);

}

}

StudentController.java Code:

**package** MVCPatternExample;

**public** **class** StudentController {

**private** Student model;

**private** StudentView view;

**public** StudentController(Student model, StudentView view) {

**this**.model = model;

**this**.view = view;

}

**public** **void** setStudentName(String name) { model.setName(name); }

**public** String getStudentName() { **return** model.getName(); }

**public** **void** setStudentId(**int** id) { model.setId(id); }

**public** **int** getStudentId() { **return** model.getId(); }

**public** **void** setStudentGrade(**double** grade) { model.setGrade(grade); }

**public** **double** getStudentGrade() { **return** model.getGrade(); }

**public** **void** updateView() {

view.displayStudentDetails(model.getName(), model.getId(), model.getGrade());

}

}

Main.java Code:

**package** MVCPatternExample;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Student model = **new** Student("Alice", 1001, 85.5);

StudentView view = **new** StudentView();

StudentController controller = **new** StudentController(model, view);

controller.updateView();

controller.setStudentName("Bob");

controller.setStudentGrade(92.0);

System.***out***.println("\n-- After update --");

controller.updateView();

}

}

Expected Output:

Student Details:

Name: Alice

ID: 1001

Grade: 85.5

-- After update --

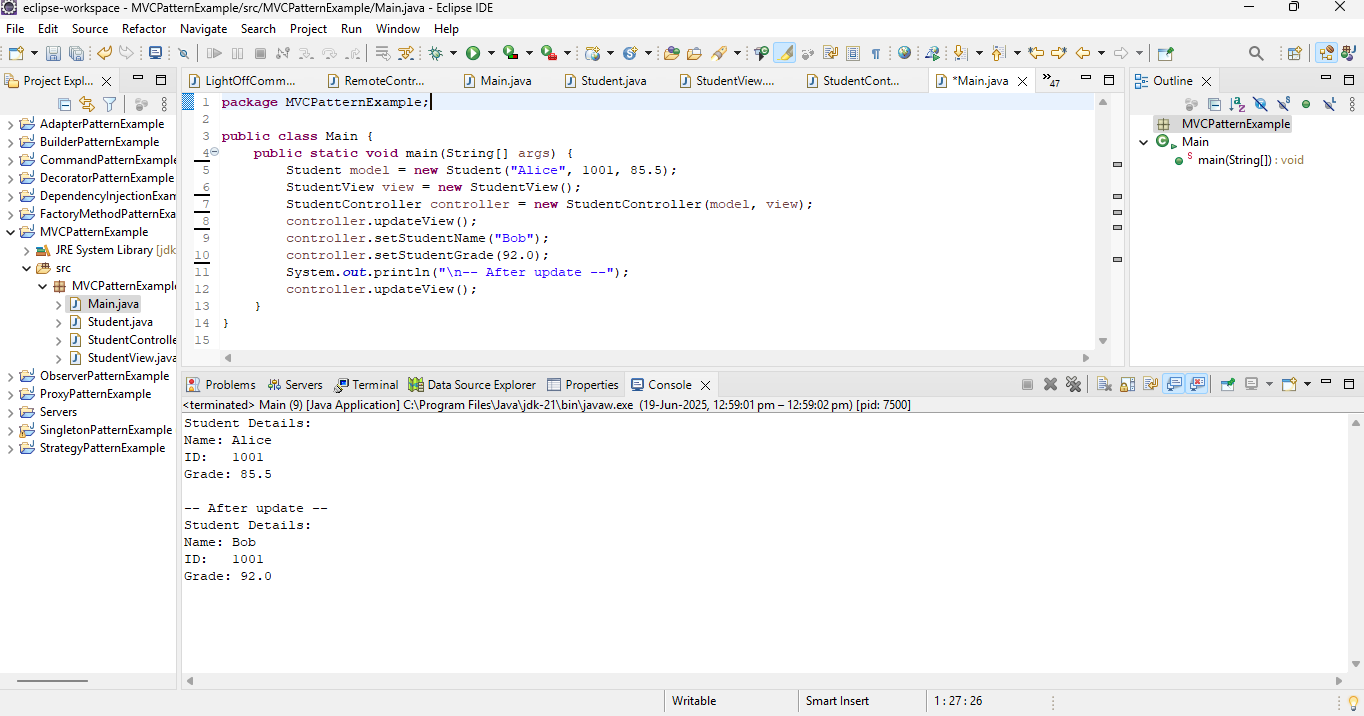
Student Details:

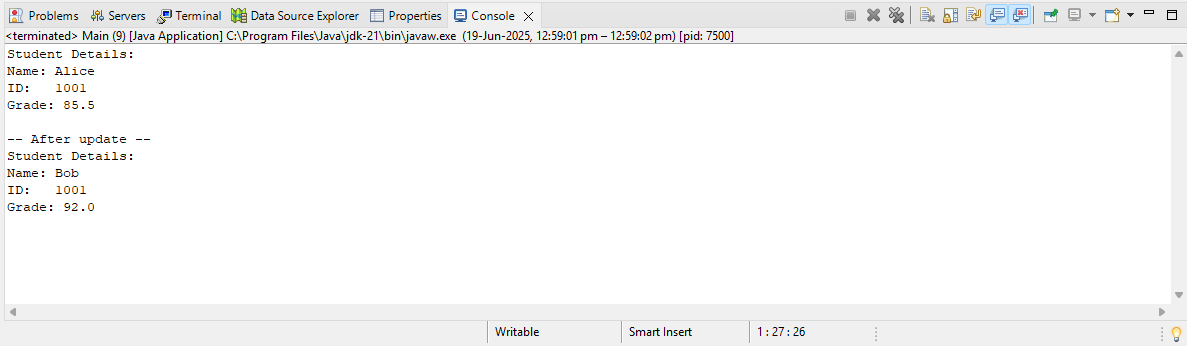
Name: Bob

ID: 1001

Grade: 92.0

Output:





**Exercise 11: Implementing Dependency Injection**

**Project Name: DependencyInjectionExample**

CustomerRepository.java Code:

**package** DependencyInjectionExample;

**public** **interface** CustomerRepository {

String findCustomerById(String id);

}

CustomerRepositoryImpl.java Code:

**package** DependencyInjectionExample;

**public** **class** CustomerRepositoryImpl **implements** CustomerRepository {

**public** String findCustomerById(String id) {

**return** "Customer[id=" + id + ", name=John Doe]";

}

}

CustomerService.java Code:

**package** DependencyInjectionExample;

**public** **class** CustomerService {

**private** CustomerRepository customerRepository;

**public** CustomerService(CustomerRepository customerRepository) {

**this**.customerRepository = customerRepository;

}

**public** **void** showCustomer(String id) {

String customer = customerRepository.findCustomerById(id);

System.***out***.println("Found: " + customer);

}

}

Main.java Code:

**package** DependencyInjectionExample;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

CustomerRepository repository = **new** CustomerRepositoryImpl();

CustomerService service = **new** CustomerService(repository);

service.showCustomer("C001");

}

}

Expected Output:

Found: Customer[id=C001, name=John Doe]

Output:

